เปิดโลก OOP
ตอนที่ ๑

รูปแบบของการเขียนโปรแกรม
และวิธีการใช้งาน
หลักการทำงาน
และการนำไปพัฒนาโปรแกรม

Object-Oriented Programming หรือ OOP คำนี้กำลังเป็นที่นิยมกันมากขึ้นทุกทีในโลกของภาษาคอมพิวเตอร์ มันคืออะไร ? จะขอเริ่มจากคำว่าออปเจ็กต์โอเรียนเต็ด (Object-Oriented) หรือ โอโอ (OO) เป็นวิธีการหรือแนวความคิดที่ไม่ได้จำกัดอยู่เพียงแค่การเขียนภาษาใดภาษาหนึ่ง ออปเจ็กต์โอเรียนต์จะไม่เน้นวิธีการหรือโพรซีเดอร์เหมือนแนวความคิดทั่วไปที่เราคุ้นเคย แต่จะเน้นที่ข้อมูล (data) มากขึ้น รวมข้อมูลและวิธีการสำหรับข้อมูลนั้น ๆ เข้าด้วยกันเป็นก้อน ๆ เดียว คล้ายกับเป็นวัตถุ (object) หนึ่ง ซึ่งวัตถุอย่างนี้ บางสำนักก็เรียกว่าเอ็นติตี้ (entity) (Bailin, 1989 และ Korson and McGregor, 1990) หรือบางพวกก็เรียกว่าคลาส (class) (Meyer, 1988)

ประวัติ

ที่จริงแล้ว ความคิดแนวแบบออปเจ็กต์โอเรียนต์ ได้เริ่มต้นมาตั้งแต่ปี ค.ศ. 1967 แล้ว โดย Ole-Johan Dahl และ Kristen Nygaard ชาวนอร์เวย์ได้คิดภาษาซิมูล่า 67 (Simula67) ที่มักถูกใช้สำหรับการจำลอง อย่างไรก็ตามภาษานี้ได้มีอิทธิพลอย่างมากต่อภาษาที่เป็นออปเจ็กต์โอเรียนต์ ทั้งหลายในยุคต่อ ๆ มาได้แก่สมอลล์ทอล์คแต่ดูเหมือนว่าออปเจ็กต์โอเรียนต์ จะยังไม่เป็นที่นิยมกันนักในช่วงแรกนี้ โดยเฉพาะในเมืองไทยน้อยคนนักที่จะรู้จักศัพท์คำนี้ จนกระทั่งภาษาซีได้ถือกำเนิดมาและเป็นที่นิยมกันอย่างแพร่หลายมาก เนื่องจากความสามารถในด้านโปรแกรมระบบ C++ จึงถือกำเนิดมา โดยมีพื้นฐานส่วนหนึ่งมาจาก C ผู้คิดก็คือ Bjarne Stroustrup จากเอทีแอนด์ที (AT&T) และก็มีออปเจ็กตีฟซี (Objective-C) โดย Brad Cox จากสต็บสโตนคอปอเรชัน (Stepstone Corporation) ช่วงหลังมาจะมีภาษา Eiffel ของ Bertrand Meyer ด้วย ซึ่งทั้ง Eiffel และ Smalltalk จัดว่าเป็นออปเจ็กต์โอเรียนต์ เต็มตัวตัวเลยเพราะไม่สามารถเขียนตามแนว procedural lan-

HISTORY OF OBJECT-ORIENTED LANGUAGES

1967	Simula	Dahl and Nygaard
early 1980s	Smalltalk	Kay, Goldberg, Ingalls
late 1980s	C++	Stroustrup
	Objective-C	Cox
	Eiffel	Meyer

รูปที่ 1 ภาษา, ชื่อผู้คิด และปีที่คิด

guage อย่างที่ถูกผสมจาก C เช่น C++ ทำได้

หลักการ

หลักการเขียนโปรแกรมตามแนวออปเจ็กต์โอเรียนต์นี้ นอกจากจะต่างกับหลักการเขียนโปรแกรมตามแบบเก่า ๆ ซึ่งเน้นวิธีการ (procedure) แต่กลับมาเน้นที่ข้อมูลจนเรียกวิธีเก่าว่า Procedural Programming และเรียกวิธีตามแบบออปเจ็กต์โอเรียนต์ว่าออปเจ็กต์โอเรียนเต็ดโปรแกรมมิงหรือโอโอพีนั่นเอง

เมื่อพิจารณาตามหลักการวิเคราะห์ระบบแบบโครงสร้าง (structured systems analysis) ซึ่งภาษาส่วนมากจะเป็นในลักษณะของการแตกตัวจากบนลงล่าง กล่าวคือ ระบบจะถูกสร้างขึ้นจากองค์ประกอบโดยรวมหรือภาพรวมทั้งหมด แล้วค่อย ๆ พิจารณาแบ่งย่อยเป็นส่วน ๆ เล็กลงมาเป็นชั้น ๆ จนกระทั่งเป็นองค์ประกอบที่เล็กสุด ดังแสดงในรูปที่ 2.1 แต่การออกแบบตามหลักออปเจ็กต์โอเรียนต์ จะเป็นทั้งแบบบนลงล่างและทั้งล่างขึ้นบนผสมกัน โดยส่วนใหญ่จะเป็นแบบล่างขึ้นบน ซึ่งจะเป็นแบบที่เริ่มวิเคราะห์และสร้างจากส่วนเล็ก ๆ หลายส่วนและรวมตัวขึ้นมาตามแบบในรูปที่ 2.2

หัวใจของออปเจ็กต์โอเรียนต์

ออปเจ็กต์โอเรียนต์ นี้มีหัวใจหรือรากฐานอยู่ 3 ประการ คือ

1. การซ่อนรูปของข้อมูล (Encapsulation) (Information Hiding)
2. Classification (Abstract Data Types)
3. Polymorphism กับ Inheritance

Encapsulation

หมายถึงการรวมข้อมูลและกระบวนการที่เกิดขึ้นกับข้อมูลนั้นเข้าด้วยกัน รวมเข้าเป็นหนึ่งเดียว ลองดูรอบตัวเราจะพบว่าสิ่งของจะเป็นตัวแสดงคุณสมบัติและการกระทำจะแสดงพฤติกรรม เช่น ลมพัด, นกร้อง ลมและนกเป็นสิ่งของที่กระทำการพัดและร้องตามลำดับ ออปเจ็กต์โอเรียนต์จะรวมคุณสมบัติของสิ่งของและพฤติกรรมของการกระทำเข้าด้วยกัน

นอกจากการรวมแล้วยังมีการซ่อนข้อมูลบางอย่างไว้ด้วย ยกตัวอย่างเช่น นก เมื่อเรามองจากภายนอกก็เห็นมันร้องโดยไม่ต้องรู้ว่ามันร้องได้อย่างไร ใช้วิธีการอะไรบ้างโดยละเอียด ในการทำให้ได้ซึ่งพฤติกรรมคือการร้อง ออกมา

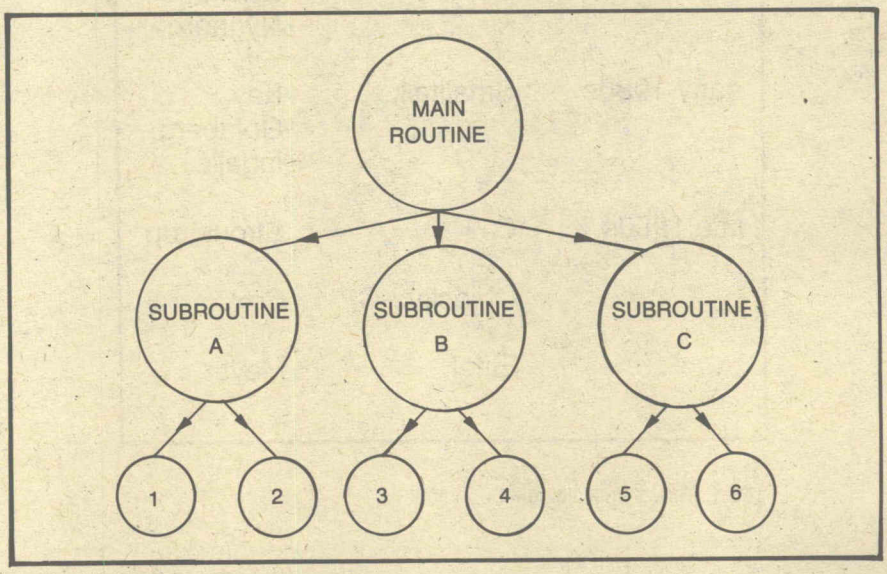
Thomas (1989) กล่าวว่า "Encapsulation เป็นศัพท์ทางเทคนิคของคำว่า Information Hiding" แต่บางคนบอกว่า 2 อย่างนี้มันไม่เหมือนกันนัก คือ การรวมตัวหรือ Encapsulation ของข้อมูลและการกระทำไม่ได้กำหนดว่า ข้อมูลภายในนี้ภายนอกจะมองไม่เห็น ในทางตรงกันข้าม หากมีการซ่อนรูปข้อมูลแล้ว นั่นหมายถึง จะต้องมีการ Encapsulation ด้วย

Bjarne Stroustrup ผู้สร้าง C++ กล่าวในวารสาร IEEE Software ฉบับเดือนพฤษภาคม 2531 ว่า "คิดโมดูลที่ต้องการ แล้วแบ่งโปรแกรมเพื่อให้ข้อมูลซ่อนอยู่ในโมดูลนั้น" ดังรูปที่ 3

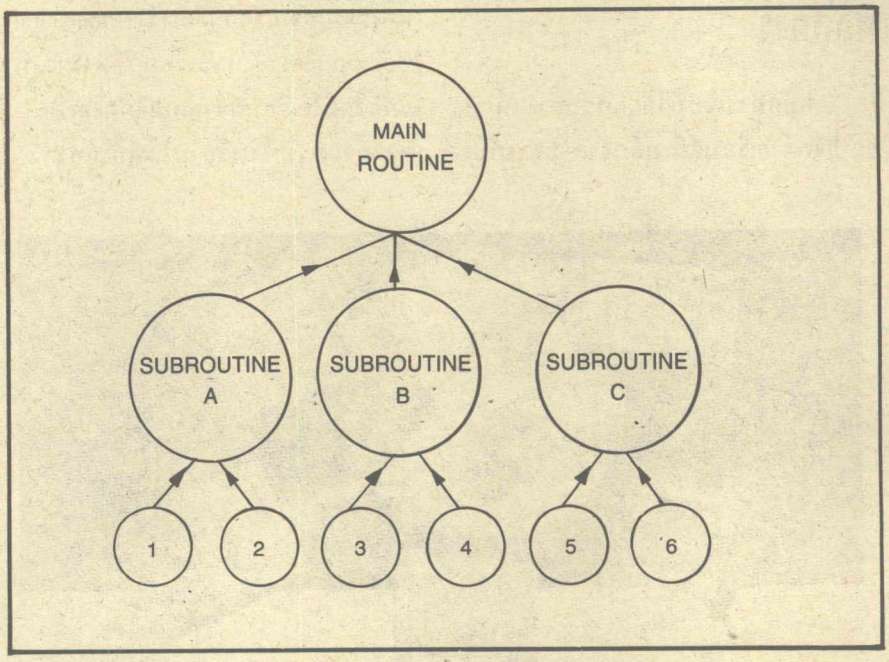
การจำแนก (Classification)

Stroustrup กล่าวว่า "คิดสร้างข้อมูลที่คุณต้องการและสร้างวิธีการกระทำทั้งหมดที่ควรมีสำหรับข้อมูลเหล่านั้น" (รูปที่ 3)ประโยคแรกก็เหมือนกับในภาษาที่ไม่ใช้ OOPL ส่วนใหญ่ ซึ่งสามารถให้โปรแกรมเมอร์กำหนดชนิดข้อมูลใหม่ขึ้นมา หรือถ้าเป็นศัพท์หน่อยก็จะเรียกว่า Abstract Data Type (ADT) เช่นเราสามารถรวบรวมข้อมูลกลายเป็น struct ในภาษาซีและกลายเป็น record ในภาษาปาสคาล แต่ใน OOPL แล้ว กระบวนการหรือ procedural หรือ function ก็สามารถถูกรวมเข้าไปด้วย โดยใน C++ จะเรียกว่าข้อมูลชนิด 'class' นั่นคือ class เท่ากับ struct + function เรียกข้อมูลหรือตัวตนที่เกิดขึ้นจากคลาสนี้ว่า 'object' เรียกส่วนประกอบย่อยของ 'object' ว่า 'member' และเรียกส่วนที่แสดงพฤติกรรมว่า 'member function' หรือ 'method'

สำหรับในเทอร์โบปาสคาลเมื่อรวมข้อมูลเข้ากับกระบวนการแล้ว จะเรียกว่าข้อมูลชนิด object type นั่นก็คือ 'object type' = record + procedure หรือ function เรียกตัวแปรที่เกิดจาก 'object type' ว่า 'instance' ส่วนประกอบย่อยของ instance คือ field และ



รูปที่ 2.1 หลักการโปรแกรมแบบบนลงล่าง (Top-down design)



รูปที่ 2.2 หลักการโปรแกรมแบบล่างขึ้นบน (Bottom-Up design)

Object-oriented philosophy builds on:

(i) Encapsulation (Data Hiding)

"Decide which modules you want; partition the program so that data are hidden in modules."

(ii) Abstract Data Types

"Decide which types you want; provide a full set of operations for each type."

Object-Oriented

"Decide which classes you want; . . . make commonality explicit by using **inheritance.**"

(Stroustrup, May 1988, IEEE Software)

รูปที่ 3 หลักการสำคัญของออปเจ็กต์โอเรียนต์

เรียกพฤติกรรมว่า method ดังสรุปศัพท์ต่าง ๆ เหล่านี้ไว้ในตารางที่ 1

การถ่ายทอดคุณสมบัติ (Inheritance)

การถ่ายทอดคุณสมบัติเป็นสิ่งที่ค่อนข้างใหม่ ที่มีใน OOPL เรื่องของ Encapsulation ภาษา Ada และ Modula-2 ได้ชื่อว่าเข้มงวดในเรื่องนี้พอสมควรในขณะที่เรื่องของ ADT ก็พอพบได้ตั้งแต่ record หรือ struct ในภาษาปาสคาลและภาษาซีตามลำดับอยู่แล้ว คุณสมบัตินี้หลายคนถือว่าเป็นประตูที่สำคัญที่จะถือว่าเป็น OOPL จริง ๆ จากในรูปที่ 3 จะใช้คำว่า Object-Oriented แทนคำว่า (iii) Inheritance เลยทีเดียว

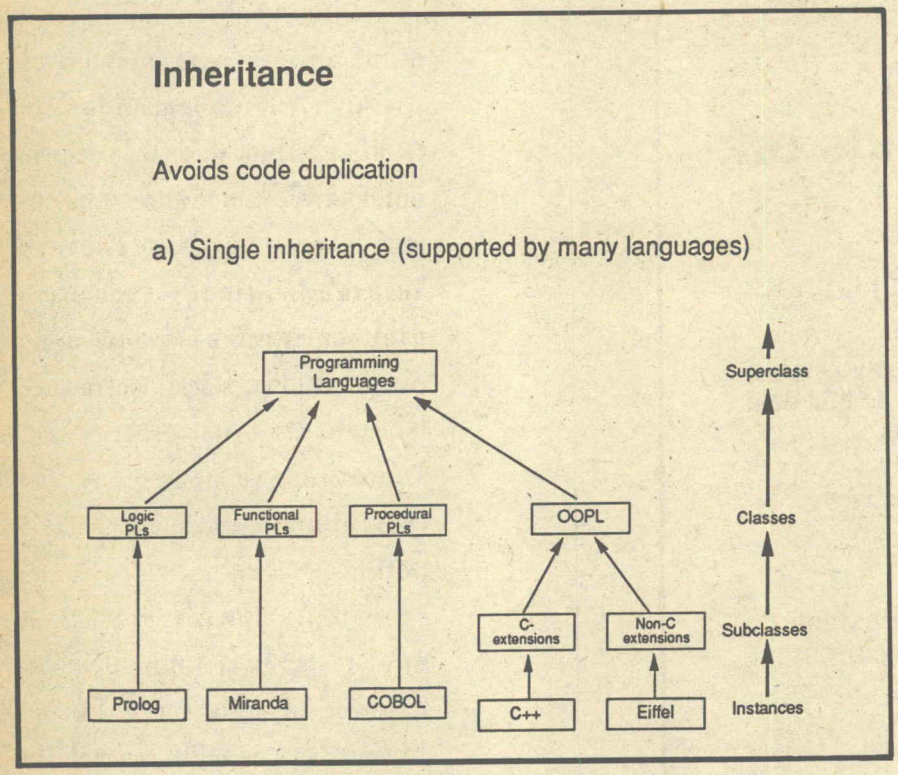
การถ่ายทอดคุณสมบัตินี้ ก็คล้าย ๆ กับในสิ่งมีชีวิตซึ่งมีการถ่ายทอดลักษณะทางพันธุกรรมจากพ่อและแม่สู่ลูก ใน OOPL ส่วนใหญ่ได้แก่ โอโอปาสคาล ออปเจ็กตีฟซี, สมอลล์ทอล์ค และแอคเตอร์ (Actor) (ซึ่งเป็นโอโอพีสำหรับไมโครซอฟต์วินโดวส์) จะสามารถถ่ายทอดคุณสมบัติมาได้จากทางเดียวเท่านั้น เรียกว่า single inheritource (SI) แต่ใน C++ และ Eiffel สามารถรับคุณสมบัติมาได้หลาย ๆ ทาง จึงเรียกว่าเป็นพวก multiple inheritance (MI)

รูป 4.1 แสดงตัวอย่างของ SI อันหนึ่ง จะเห็นว่า ทุกออปเจ็กต์จะมี parent ได้ไม่เกิน 1 เช่น ในภาษาโปรล็อกมี PLs ลอจิกเป็น parent, Eiffel มี Non-C extension เป็น parent เป็นต้น แต่ในรูปที่ 4.2 จะพบว่ามีไฮบริดที่มี parent อยู่ 2 ตัวคือ C-extensions และ procedural PLs ซึ่งลักษณะเช่นนี้เป็นตัวอย่างของ MI

ความสามารถของ inheritance จะช่วยให้เกิดการใช้คุณลักษณะและพฤติกรรมบางอย่างร่วมกันได้ ทำให้สร้างชนิดข้อมูลที่ซับซ้อนได้จากข้อมูลที่ง่ายกว่า โดยไม่ต้องเขียนโค้ดมากมายใหม่อีก และถ้าต้องการชนิดข้อมูลใหม่ซึ่งมีบางส่วนคล้ายกับที่มีอยู่หลังก็สา-

Concept	C++	Turbo PASCAL
data structure	struct	record
user-defined type	class	object type
Variable of user-defined type	object	instance
data field	member	field
behavior	member function, method	method

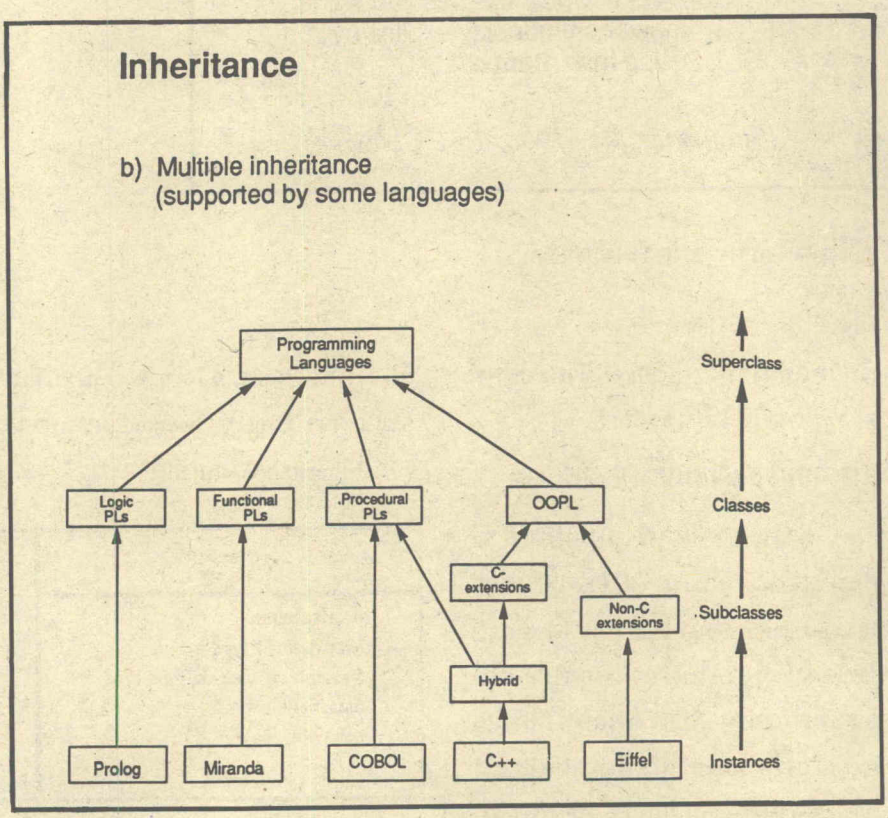
ตารางที่ 1 เปรียบเทียบ C++ กับ Turbo PASCAL



รูปที่ 4.1 ตัวอย่างแสดง single Inheritance

มารถนำที่มีอยู่มาใช้ได้โดยเพิ่มลักษณะและพฤติกรรมพิเศษบางอย่างเข้าไป

ทีนี้เมื่อเห็นอย่างนี้แล้ว คงคิดว่า MI น่าจะดีกว่า SI จริงหรือเปล่า ? อันนี้เป็นเรื่องที่นักออกแบบ OOPL ทั้งหลายถกเถียงกันอยู่ จริงอยู่ MI ลดข้อจำกัดบางอย่างของ SI ลงได้ แต่ความสับสนก็อาจเกิดได้ในกรณีที่รับคุณสมบัติหรือพฤติกรรมบางอย่างซ้ำซ้อนกัน เช่น ข้อมูลชนิดมัลติมีเดียมาจากชนิด เสียง และ รูปภาพ โดยเสียงมีคุณสมบัติของความถี่, ความดัง, และระยะเวลา มีพฤติกรรมของการ play, rescale, โหลด และสโตร์ (store) ส่วนรูปภาพมีคุณสมบัติของ ลักษณะของภาพ, ขนาด, และสี มีพฤติกรรมของการ rescale, โหลด และสโตร์ (store) ซึ่งมีชื่อเหมือนกับของเสียง ทางแก้ก็คือจะต้องบ่งบอกให้ชัด



รูปที่ 4.2 ตัวอย่างแสดง Multiple Inheritance ได้แก่ ไฮบริด (hybrid)

อีกหน่อยในกรณีเหล่านี้ว่าจะ rescale หรือโหลดหรือสโตร์เสียงหรือรูปภาพเป็นต้น

แล้ว polymorphism คืออะไร ในห้องเรียนภาษาอังกฤษ อาจารย์จะแยกศัพท์คำนี้ออกเป็น poly- แปลว่า หลาย ๆ -morph แปลว่า รูปร่าง ใช่แล้ว ศัพท์คำนี้หมายถึงการมีรูปร่างหน้าตามากกว่า 1 ลองดูจากตัวอย่างนี้ สมมติเรามีข้อมูลชนิด รถและเรือ โดย ยานพาหนะมีพฤติกรรมการขับเคลื่อน ดังนั้นทั้งรถและเรือก็สามารถมีพฤติกรรมการขับเคลื่อนด้วย แต่ว่าวิธีการที่จะทำให้เกิดพฤติกรรมนี้ จะแตกต่างกันระหว่างรถและเรือ เช่นรถต้องขับ และเรือต้องพาย เป็นต้น เมื่อเราบอกให้ยานพาหนะขับเคลื่อน มันจะพิจารณาเองตามชนิดว่าจะใช้การขับหรือพาย

1. Definition of an OOPL

Level 1 *(Object-based modular structure)*: Systems are modularized on the basis of their data structures.

Level 2 *(Data abstraction)*: Objects should be described as implementations of abstract data types.

Level 3 *(Automatic memory management)*: Unused objects should be deallocated by the underlying language system, without a programmer intervention.

Level 4 *(Classes)*: Every non simple type is a module and every high-level module is a type.

Level 5 *(Inheritance)*: A class may be defined as an extension or restriction of another.

Level 6 *(Polymorphism and dynamic binding)*: Program entities should be permitted to refer to objects or more than one class, and operations should be permitted to have different realizations in different classes.

Level 7 *(Multiple and repeated inheritance)*: It should be possible to declare a class as heir to more than one class and more than once to the same class.

B. Meyer, 1988. *Object-oriented Software Construction*

รูปที่ 5 คำจำกัดความของ OOPL ตามนิยามของ B.Meyer, 1988

ตัวอย่าง OOPL

ตามที่เกริ่นไว้ในตอนต้นถึงรายชื่อภาษาต่าง ๆ ที่เป็น OOPL ในตอนนี้ก็แนะนำให้รู้จักมากขึ้นและลึกเข้านิดหน่อย OOPL บางตัว เช่น สมอลล์ทอล์ค, แอคเตอร์ (OOPL สำหรับวินโดวส์), และ Eiffel จัดว่าเป็นโอโอพีอย่างเต็มตัว เพราะทุกอย่างจะถือว่าเป็นวัตถุ มักเรียกว่าเป็น pure OOPL เช่น สมอลล์ทอล์ค (ซึ่งเป็น OOPL ระดับรุ่นปู่ทีเดียว รอง ๆ มาจากซิมูล่า-67 พัฒนามาจาก Xerox's Palo Alto Research Center) แม้แต่การบวกเลขง่าย ๆ ยังมองเป็นออปเจ็กต์โอเรียนต์เลย โดยการส่งคำสั่งบอกให้แก่ออปเจ็กต์ของตัวเลขเพื่อทำการบวก หรืออย่างการเปรียบเทียบที่ปกติเราจะพบประโยค if ถ้าเป็นในสมอลล์ทอล์คจะทำการส่งคำสั่ง IfTrue ไปให้ออปเจ็กต์ Condition ที่จะทำการประเมินเปรียบเทียบออกมาเป็นจริงหรือเท็จ อ่อนหน่อยก็เป็น แอคเตอร์ และ Eiffel ซึ่งมีขนมหวานให้แก่โปรแกรมเมอร์ที่ยังไม่คุ้นเคยออปเจ็กต์โอเรียนต์เท่าไรด้วย แต่ขณะเดียวกัน ก็ยังคงความเข้มข้นของออปเจ็กต์โอเรียนต์ไว้อย่างเต็มที่

อีกพวกหนึ่ง เช่น C++ และเทอร์โบปาสคาลพวกนี้จัดเป็นกลุ่มลูกผสม เป็นภาษาที่มีรากฐานมาภาษาโครงสร้างทั่วไปมาเติมคุณสมบัติทางออปเจ็กต์-โอเรียนต์เข้าไปทำให้อาจจะเป็น OOPL ที่นิยมมากที่สุดกลุ่มหนึ่ง เพราะเป็น OOPL ที่มีความสะดวกสบายตามสไตล์ของภาษาโครงสร้างทั่วไป คือ C++ สามารถนำโค้ด C ธรรมดามาคอมพายล์และรันได้ และโอโอเทอร์โบปาสคาลก็ทำ Structured Pascal ปกติมาเพิ่มความสามารถทางออปเจ็กต์โอเรียนต์

นอกจากนี้ ยังมีออปเจ็กตีฟซีที่ Brad Cox เป็นผู้ออกแบบ เขาได้สร้างศัพท์คำว่า "Software IC" หรือ "ไอซีที่เป็นซอฟต์แวร์" ขึ้นมาก โดยใช้เรียกแก่ออปเจ็กต์ที่ถูกสร้างขึ้นและถูก encapsulated อย่างเรียบร้อยสามารถนำมาใช้ใหม่ได้ง่าย เหมือนเป็น ไอซีสำเร็จรูปอะไรอย่างนั้น

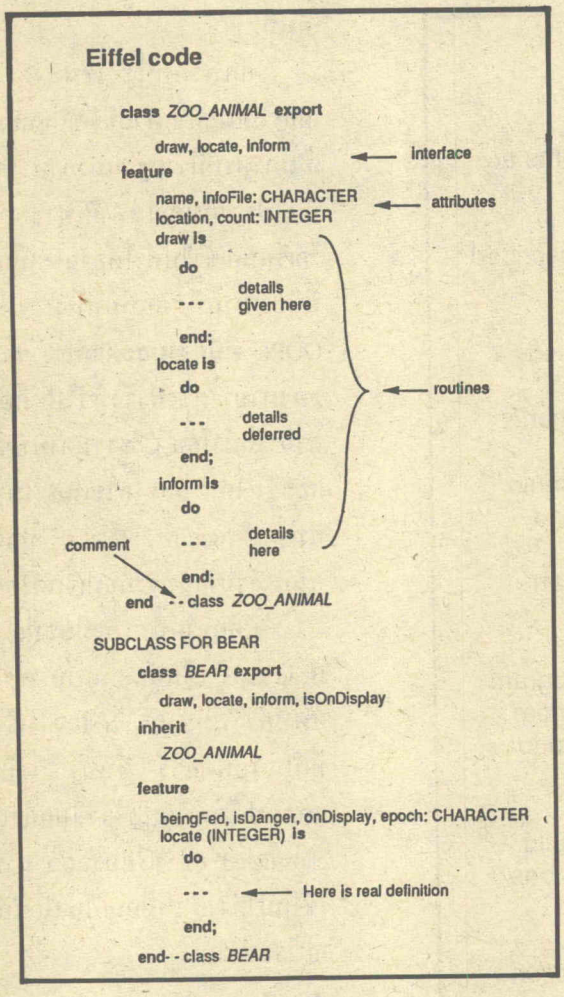
ว่าเป็นหลักการ

หลังจากแนะนำ OOPL หลาย ๆ ตัวไปบ้างแล้ว ตอนนี้ลองมาแบ่งระดับความเป็นออปเจ็กต์โอเรียนต์ตามนิยมที่ Meyer ผู้คิดภาษา Eiffel ได้กำหนดแบ่งภาษาไว้ 7 ระดับ ตามรูปที่ 5 คือ

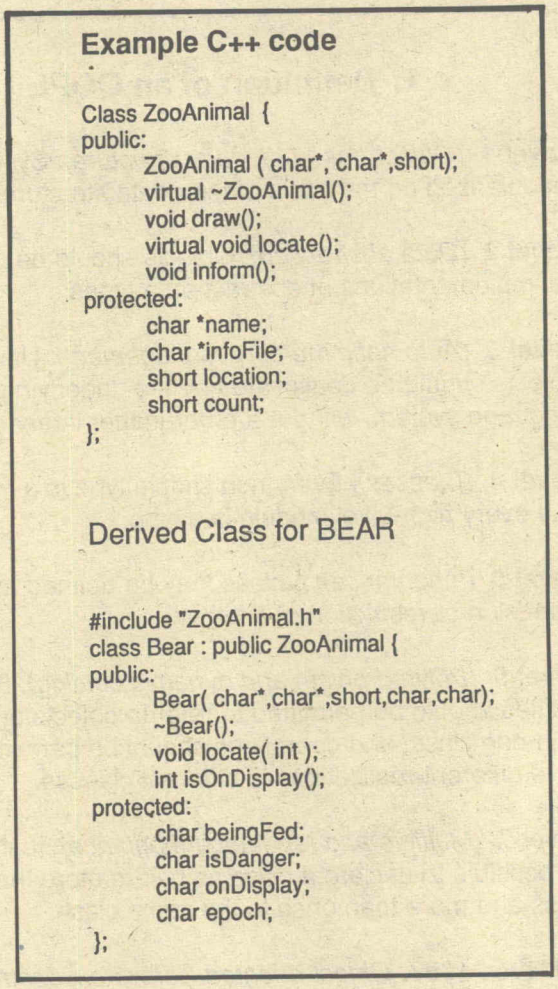
ระดับที่ 1 object-based modular structure ภาษาระดับ 3GL หลายภาษา เช่นฟอร์แทรนถ้าเขียนิสับรูทีนที่เป็นโมดูลดี ๆ ก็จัดอยู่ในข้อนี้ได้

ระดับที่ 2 Data Abstraction ภาษาระดับ 3GL หลาย ๆ ภาษาก็มีคุณสมบัติของ ADT ซึ่งได้อธิบายไว้แล้วช่วงต้น

ระดับที่ 3 Automatic Memory Management หรือ Automatic Garbage collection หมายถึงจะต้องสามารถจัดการ



รูปที่ 6.1 ตัวอย่างการเขียนคลาสและการถ่ายทอดคุณสมบัติ (Inheritance) ในภาษา Eiffel



รูปที่ 6.2 ตัวอย่างการเขียนคลาสและการถ่ายทอดคุณสมบัติ (Inheritance) ในภาษา C++

หน่วยความจำได้ดี เช่น คืนหน่วยความจำได้เองอัตโนมัติ เมื่อไม่ต้องใช้อย่างเวลาที่เขียน C การขอ malloc แก่ตัวแปรโลคอล เมื่อจะออกจากฟังก์ชันนั้น โดยไม่ใช่หน่วยความจำนั้นอีก ก็ควรจะคืนให้แก่ระบบเองเป็นต้น ซึ่งช่วยลดปัญหาหน่วยความจำหมดขณะรันโปรแกรมอยู่ และสะดวกแก่โปรแกรมเมอร์ อย่างในภาษา Eiffel จะมีตัวเลือกในไฟล์สำหรับตั้งค่าระบบ ส่วนในภาษาสมอลล์ทอล์ค การคืนหน่วยความจำโดยอัตโนมัติเช่นนี้ถือเป็นเรื่องปกติ ในระหว่างที่รัน มันจะขอหน่วยความจำจากระบบเมื่อเห็นว่าซีพียูมีงานน้อย และจะคอยคืนหน่วยความจำบริเวณที่เลิกใช้แล้วให้ตลอดเวลา โดยไม่รอให้หน่วยความจำหมด แล้วจึงค่อยมารวบรวมสะสางขยะ ซึ่งฟังดูก็ดี แต่อาจจะไม่เหมาะกับงานที่ต้องการความแม่นยำสูงในเรื่องการจัดเวลา ความสามารถในการจัดการหน่วยความจำลักษณะอย่างนี้ยังไม่มีใน C++ หรือออปเจ็กติฟซี

ระดับที่ 4 คลาส (Classes) ถึงระดับนี้ก็ต้องพร้อมถึงคุณสมบัติเกี่ยวกับ ADT ที่กล่าวถึงในตอนต้น มีเพียงไม่กี่ภาษาเท่านั้นที่จัดอยู่ในระดับนี้ นอกจากนั้น Meyer ยังเสริมอีกว่า ชนิดข้อมูลที่ไม่ใช่ข้อมูลพื้นฐานอย่าง int, char. จะต้องเป็นโมดูลและทุกโมดูลจะต้องมีชนิดใดชนิดหนึ่งเช่น ถ้าเราสร้างข้อมูลชนิด Customer ขึ้นมาเราจะต้องเขียนรูทีนที่เกี่ยวข้องกับ Customer เข้ามาให้เป็นโมดูลเป็นต้น

ระดับที่ 5 อินเฮอรแตนซ์ (Inheritance) เป็นคุณสมบัติที่หลายคนบอกว่า OOPL จะต้องมีถึงจะเป็น OOP ได้ ตัวอย่างในรูป 6 แสดงเปรียบเทียบการเขียนคลาสที่มีการถ่ายทอดคุณสมบัติระหว่าง Eiffel และ C++(รูปที่ 6.1

และ 6.2 ตามลำดับ)

ในรูป 6.1 แสดงการกำหนดคลาส Zoo_Animal จะมีคำว่า Export เพื่อบอกส่วนที่จะให้ข้างนอกติดต่อเข้ามาได้ ได้แก่ draw, locate, inform รายละเอียดเหล่านี้กำหนดไว้ในส่วน feature ซึ่งจะประกาศตัวแปรที่ซ่อนไว้คือ ภายนอกมองไม่เห็นเช่น name, count เป็นต้น และประกาศวิธีการสำหรับส่วนที่ประกาศเป็น export โดยใช้คำว่า is, do, end ส่วนล่างเป็นคลาส bear ซึ่งรับการถ่ายทอดมาจาก Zoo_Animal

ในรูป 6.2 เป็นตัวอย่างการประกาศคลาสของ C++ ซึ่งใช้เหมือนกับ Eiffel ในรูป 6.1 คลาสของ C++ จะแบ่งเป็น 3 ส่วนคือ public, protected และส่วน private ซึ่งไม่ได้แสดงให้เห็นในที่นี้ ฟังก์ชันสมาชิก จะสามารถเขียนไว้ได้ทั้งภายในคลาสเลย หรือเป็นแบบ inline เลยก็ได้ นอกจากนั้นแถมยังเขียนอยู่คนละไฟล์เลยก็ยังได้ มีให้เลือกตั้งหลายค่า ซึ่งขึ้นอยู่กับปัจจัยต่าง ๆ กัน เช่น ถ้าเราใส่คำว่า inline เข้าไป คอมพายเลอร์ก็จะใส่โค้ดส่วนนี้เข้าไปเลย โดยไม่ต้องมีการผ่านค่าเรียกฟังก์ชัน ซึ่งเหมาะกับงานสั้น ที่ต้องการความเร็วสูง เป็นต้น ในส่วนล่างจะแสดงการถ่ายทอดคุณสมบัติจาก ZooAnimal ไปให้คลาส Bear

ระดับที่ 6 Polymorphism and dynamic binding อย่างที่เคยอธิบายไว้แล้ว คือ วัตถุจะต้องสามารถถูกอ้างได้มากกว่า 1 คลาส และมีกระบวนการจัดการข้อมูลต่างจากคลาสได้ขณะรันโปรแกรม เช่น เราเรียกรูทีน print โดยส่งคำสั่งหรือข่าวสาร (message) ให้เป็นวงกลมหรือตัวหนังสือก็ได้ และมันจะต้องแสดงผลเป็นการวาดรูปวงกลม

```
void show(obj)
struct object *obj;
{
        switch(obj-<object_type)
        {
            case POINT:
                showpoint(obj);
                break;
            case CIRCLE:
                showcircle(obj);
                break;
            default:
                printerror("Incorrect object type");
        }
}
```

```
void show(obj)
struct object *obj;
{
        obj.show;
}
```

รูปที่ 7 ตัวอย่างเปรียบเทียบการมีคุณสมบัติ Polymorphism ในด้านขวา กับการไม่มีในด้านซ้าย

หรือแสดงอักษรออกมาได้อย่างถูกต้อง ลักษณะนี้เรียกอีกอย่างหนึ่งว่าการโอเวอร์โหลด (overload) ฟังก์ชัน

ประโยชน์ของการทำ polymorphism ดูได้จากตัวอย่าง เช่น การจะแสดงวัตถุหนึ่งที่เป็นจุดหรือวงกลมในภาษา C ซึ่งไม่มีคุณสมบัติของ polymorphism จะต้องใช้สวิตช์ในขณะที่ใน C++ ข้อมูลจุดและวงกลมซึ่งรับการถ่ายทอดคุณสมบัติจากออปเจ็กต์จะตอบสนองต่อคำสั่งหรือข่าวสาร (message) จะพิจารณาเองว่าเป็นจุดหรือวงกลม ดังตัวอย่างเปรียบเทียบกันในรูป 7 นอกจากนี้ถ้ามีมากขึ้นเช่น เส้นตรง, สี่เหลี่ยม, หรือรูปหลายเหลี่ยม เป็นต้น ในภาษาที่ไม่มี polymorphism จะต้องการ case มากขึ้น เช่นใน C อย่างน้อยก็ 3 บรรทัดใน 1 ชนิด โดยไม่ต้องเพิ่มเติมอะไรใน C++ เลย

หลักการอีกอันหนึ่งที่เกี่ยวข้องกับ polymorphism นอกจากการโอเวอร์โหลดฟังก์ชันก็คือเรื่องของการโอเวอร์โหลดโอเปอเรเตอร์ (overload operator) หมายถึงการใช้สัญลักษณ์อันเดียวกันโดยมีความหมายและกระบวนการที่ต่างกัน เช่น A=B+C เครื่องหมาย + นี้จะหมายถึงสิ่งที่ต่างกันถ้า B และ C มีชนิดเปลี่ยนไปเช่นเป็นเลขจำนวนเต็มหรือว่าเป็นเลขเชิงซ้อน ใน OOPL คุณสามารถสร้างความหมายให้แก่เครื่องหมายต่าง ๆ ได้เช่น + อาจจะสร้างขึ้นสำหรับการบวกแมตริกซ์ หรือโต๊ะก็ยังได้ โดยขึ้นอยู่กับคุณว่าจะให้โต๊ะ + โต๊ะเป็นอะไร อาจหมายถึงการขันน็อต เพื่อต่อด้านยาวกับด้านยาวหรือว่าต่อด้านกว้างกับด้านกว้างก็ได้ตามใจ เหล่านี้ไม่สามารถพบได้ภายนอกโลกออปเจ็กต์-โอเรียนต์ ซึ่งการทำโอเวอร์โหลดโอเปอเรเตอร์จะได้เพียงเฉพาะภายในขอบเขตที่คอมพายเลอร์มีไว้ให้ แต่อย่างไรก็ตามการทำโอเปอเรเตอร์โอเวอร์โหลดได้ก็ไม่ได้มีแต่ข้อดีอย่างเดียวอาจจะเกิดอันตรายได้เหมือนกัน เพราะสามารถทำให้โค้ดที่อ่านดูรู้สึกแปลก ๆ ได้ อย่างกรณีการเอาโต๊ะมาต่อกัน เราไปใช้เครื่องหมายบวกก็ดูแปลกดีไปอย่าง

ระดับที่ 7 Multiple & repeated inheritance การถ่ายทอดคุณสมบัติลักษณะนี้ ยังเป็นที่ถกเถียงกันอยู่ว่าจำเป็นแค่ไหน มีไม่กี่ภาษาที่มีคุณสมบัตินี้ทั้งในเวอร์ชันปัจจุบันและเวอร์ชันที่กำลังจะออกมา

คุณสมบัติอื่น ๆ

นอกเหนือจากหลักการต่าง ๆ ตามที่กล่าวมาแล้ว OOPL หลาย ๆ ภาษายังมีคุณสมบัติพิเศษบางอย่างที่ไม่มีในภาษาที่ไม่ใช่ออปเจ็กต์โอเรียนต์ได้แก่

Genericity

คุณสมบัติพิเศษอย่างหนึ่งของ OOPL คือ การที่อะเรย์สามารถมีตัวชี้หรืออินเด็กซ์เป็นอะไรก็ได้นอกเหนือจากเลขจำนวนเต็มที่เราคุ้นเคยกัน อาจจะเป็นเลขจำนวนจริง, ACCOUNT หรือ COLOR ก็ได้คุณสมบัติอย่างนี้ มีในภาษา Eiffel, CLU และ ADA แต่ยังไม่มีใน C++ เวอร์ชันปัจจุบัน โดย C++ จะสามารถแก้โดยการใช้ type conversion หรือ type casting แทน

```
If amount > 0 then
        Call subroutine CALCULATE
else
        Flag error
```

รูปที่ 8.1 ตัวอย่างการตรวจสอบค่าก่อนทำงาน ซึ่งพบในภาษาส่วนมาก

```
class ACCOUNT export
        ..., deposit, ..., ...
feature
        balance;INTEGER;
        owner:STRING;
        .
        .
        .
        deposit (amount:INTEGER) is
            require
                amount > 0
            do
                balance : =balance + amount
            ensure
                balance = old balance + amount
            end;
            .
            .
            .
```

รูปที่ 8.2 คุณสมบัติการตรวจสอบค่าก่อนเข้าและออกชุดการทำงานในภาษา Eiffel

Assertion

เป็นคุณสมบัติพิเศษที่มีใน Eiffel แต่ยังไม่มีใน C++, ออปเจ็กตีฟซีหรือสมอลล์ทอล์ค สมมติเราจะเขียนคลาสของบัญชีเงินฝากอันหนึ่ง ถ้ามีสั่งส่งมาให้ทำการฝาก หากเป็นในภาษาอื่น เราต้องตรวจสอบคำของเงินที่จะฝาก เช่น ต้องมากกว่า 0 ก่อนทำการบวก โดยอาจจะเขียนเป็นโค้ดแบบในรูปที่ 8.1 เป็นต้น แต่ใน Eiffel จะมีการเขียนประโยคตรวจสอบก่อนและหลัง (Pre-

Language Comparison of OO Terminology

Smalltalk	C++	Objective-C	Object-Pascal	Eiffel	CLOS
Object	Object	Object	Object	Object	Instance
Class	Class	Factory	Object type	Class	Class
Method	Member function	Method	Method	Routine	Method Generic function
Instance variable	Member	Instance variable	Object variable	Attribute	Slots
Message	Function call	Message expression	Message	Applying a routine	Generic function
Subclass	Derived class	Subclass	Descendant type	Descendant	Subclass
Inheritance	Derivation	Inheritance	Inheritance	Inheritance	Inheritance

© Winblad *et al.* (1990)

รูปที่ 9.1 ตารางเปรียบเทียบ Terminology ของ OOPL บางภาษา

Comparison of OO Language Features

Feature	Smalltalk	C++	Eiffel	Turbo Pascal
Persistence	Some (whole system)	No	Yes	No
Multiple inheritance	No	Ver 2.0	Yes	No
Typing	Dynamic	Static	Static	Static
Binding	Dynamic	Default static Optional dynamic	Dynamic	Default static Optional dynamic
Autogarbage collection	Yes	No	Yes	No
Exception handling	No	No	Yes	No
Assertions	No	No	Yes	No
Libraries	Yes	From some vendors but mainly third party	Yes	Yes

รูปที่ 9.2 ตารางเปรียบเทียบคุณสมบัติของ OOPL บางภาษา

condition and Postcondition) ไว้ในคลาส แล้วจะได้ไม่ต้องไปเขียน IF THEN ELSE อย่างนี้อีก การเขียน Precondition ในกรณีนี้ จะตรวจสอบสอบว่าเงินฝากจะต้องเป็นเลขบวกและจะเขียนโดยใช้คำว่า 'require' และการตรวจสอบภายหลังการทำงานหรือ Postcondition จะเขียนโดยใช้คำว่า 'ensure' อย่างในรูปที่ 8.2

Exception Handling

เป็นสิ่งสำคัญในการที่จะตรวจสอบย้อนกลับในกรณีที่เกิดข้อผิดพลาดขึ้น ซึ่งพบได้ใน OOPL หลายภาษาเช่น Ada, CLU, Eiffel, สมอลล์ทอล์ค และ CLOS แต่ยังไม่มีในออปเจ็กตีฟซีและ C++ การมีภระบวนการจัดการข้อผิดพลาดนี้เมื่อเกิดความผิดพลาดขึ้นขณะรัน โปรแกรมจะหยุดและจะไปทำงานตามที่กำหนดไว้ ใน Eiffel กลไกนี้ยังสามารถเลือกได้ว่า จะให้ลองทำใหม่อีกครั้ง หรือว่าจะให้มีการกู้สถานภาพคืน (rescue) อย่างเวลาเราโทรโมเด็มแล้วสายไม่ว่างก็น่าจะรอสักพักหนึ่งแล้วโทรให้ใหม่ถ้ายังไม่ได้ก็ลองใหม่อีกในช่วงเวลาหนึ่งไม่ใช้ว่าไม่ได้ก็กลับมาเลย

บทส่งท้าย

รูปที่ 9.1 เป็นตารางสรุปคำศัพท์ที่ใช้ในภาษา OOPL กลุ่มหนึ่งซึ่งได้กล่าวถึงมาแล้วพอสมควร ซึ่งบางส่วนก็มีลงไว้แล้วในตารางในรูปที่ 4 สำหรับรูปที่ 9.2 เป็นตารางเปรียบเทียบคุณสมบัติของ OOPL 4 ภาษาที่กล่าวถึงบ่อย ๆ ในบทความนี้ คิดว่าทั้ง 2 ตารางนี้จะช่วยสรุปความเข้าใจและไว้ใช้อ้างอิงได้บ้าง